南京大学
NANJING UNIVERSITY

人工智能导论

对抗搜索
(Adversarial Search)

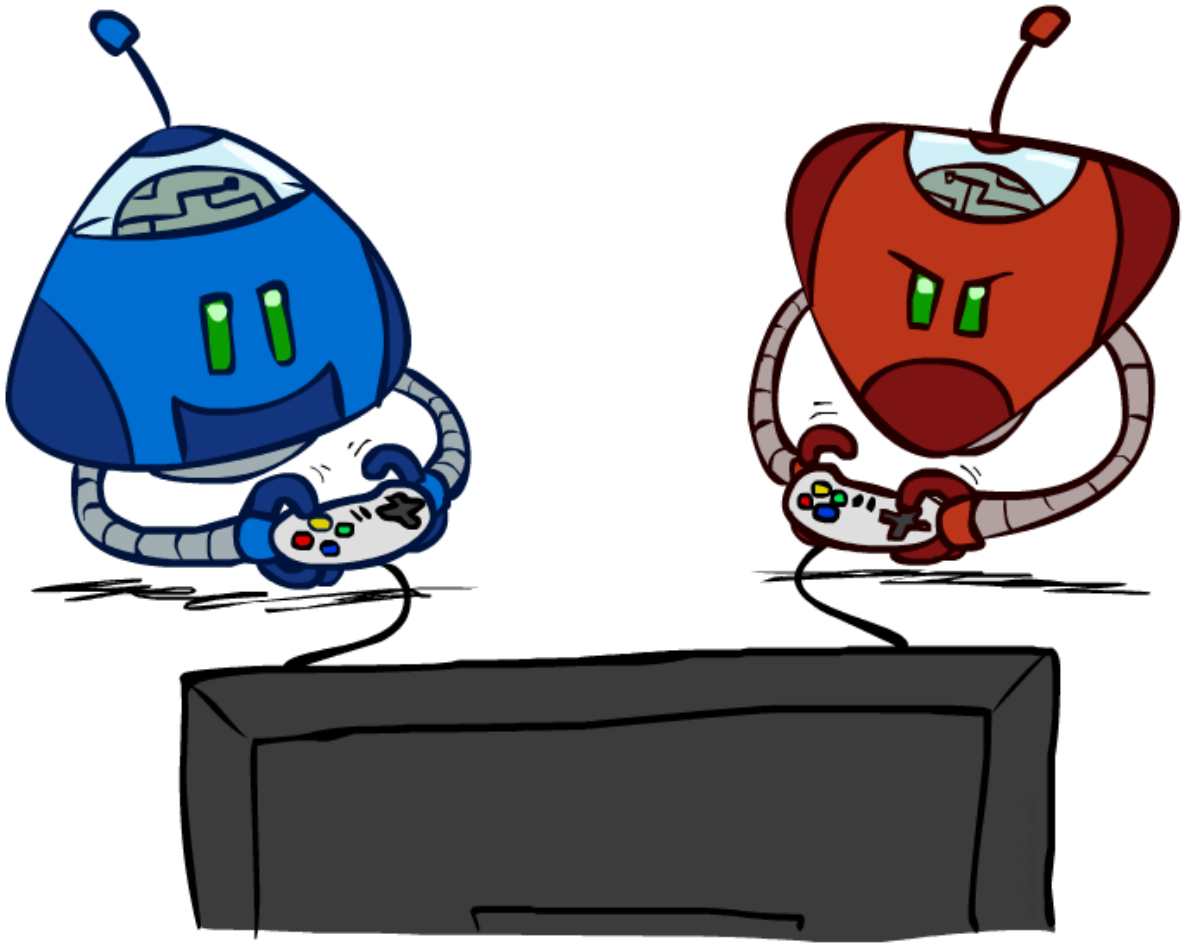
郭兰哲

南京大学 智能科学与技术学院

https://www.lamda.nju.edu.cn/guolz

Email: guolz@nju.edu.cn

对抗搜索



提纲

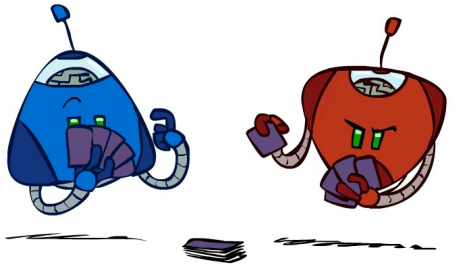
- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

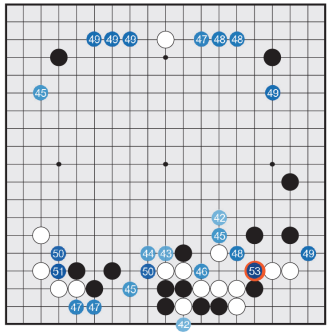
提纲

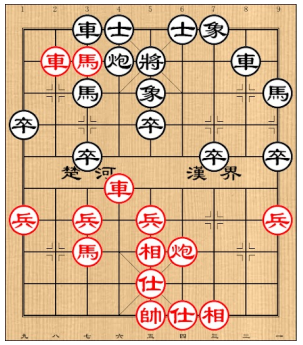
- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

博弈(Game)

对抗搜索（Adversarial Search）也称为博弈搜索（Game Search）



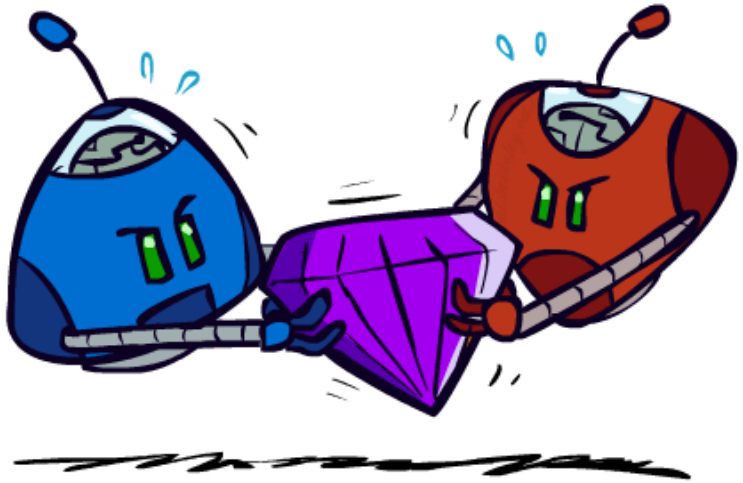




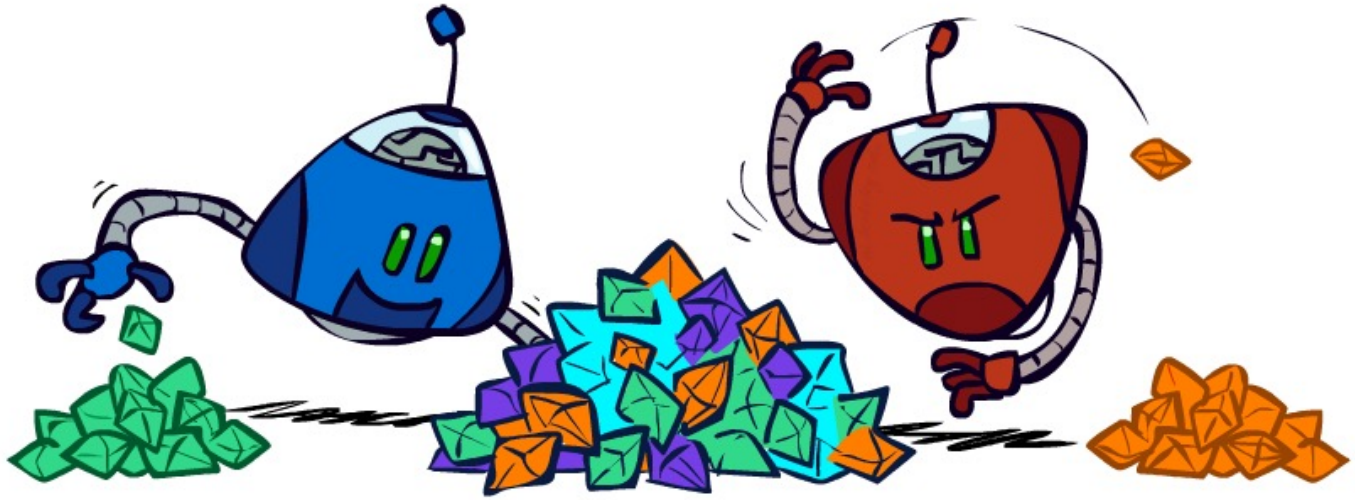
博弈的种类：

- ✓ 确定的、有随机性的
- ✓ 是否有完整信息？
- ✓ 几个玩家？
- ✓ 是不是零和博弈？

零和博弈



一个玩家赢了，则对手一定输了



你可能赚了，但我也不亏

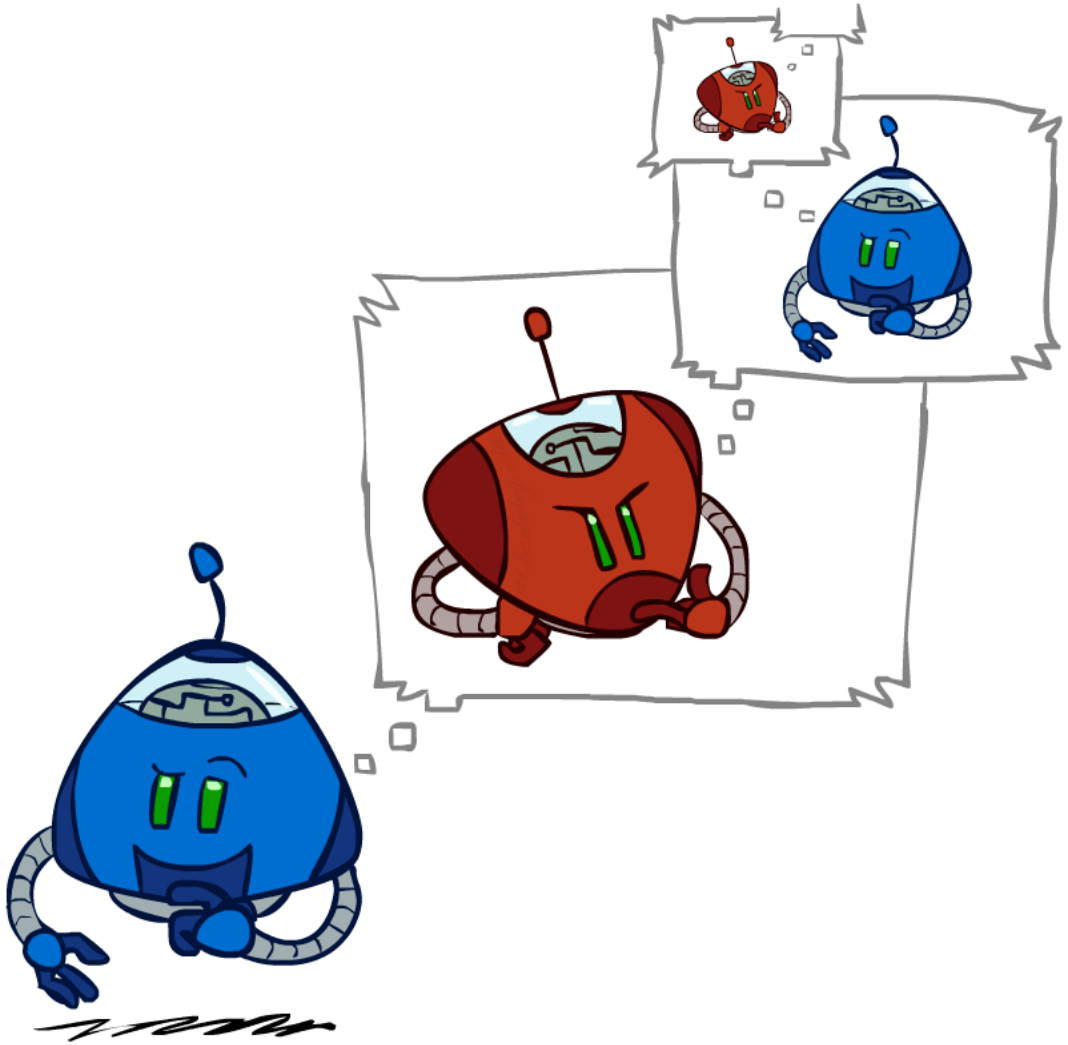
双人零和博弈

我们考虑信息确定、全局可观察、竞争对手轮流行动、输赢

收益零和假设下的双人博弈问题

VS.

对抗搜索



提纲

- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

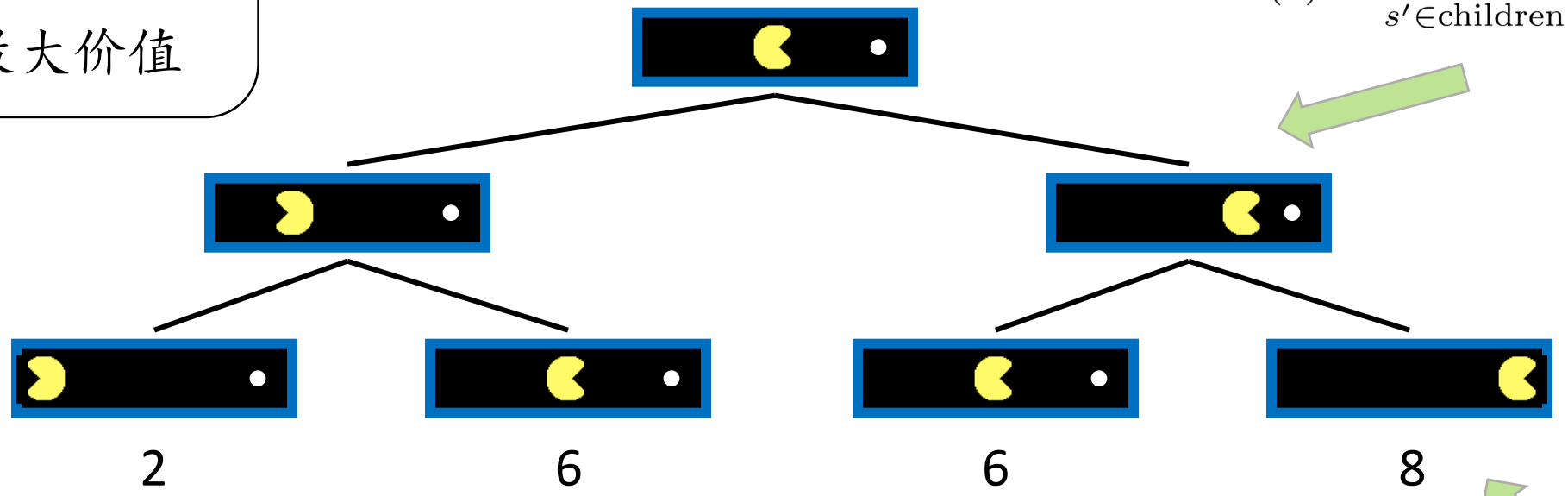
单一Agent搜索树

状态的价值 V

从当前状态出发能获得的最大价值

Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Terminal States:

$$V(s) = \text{known}$$

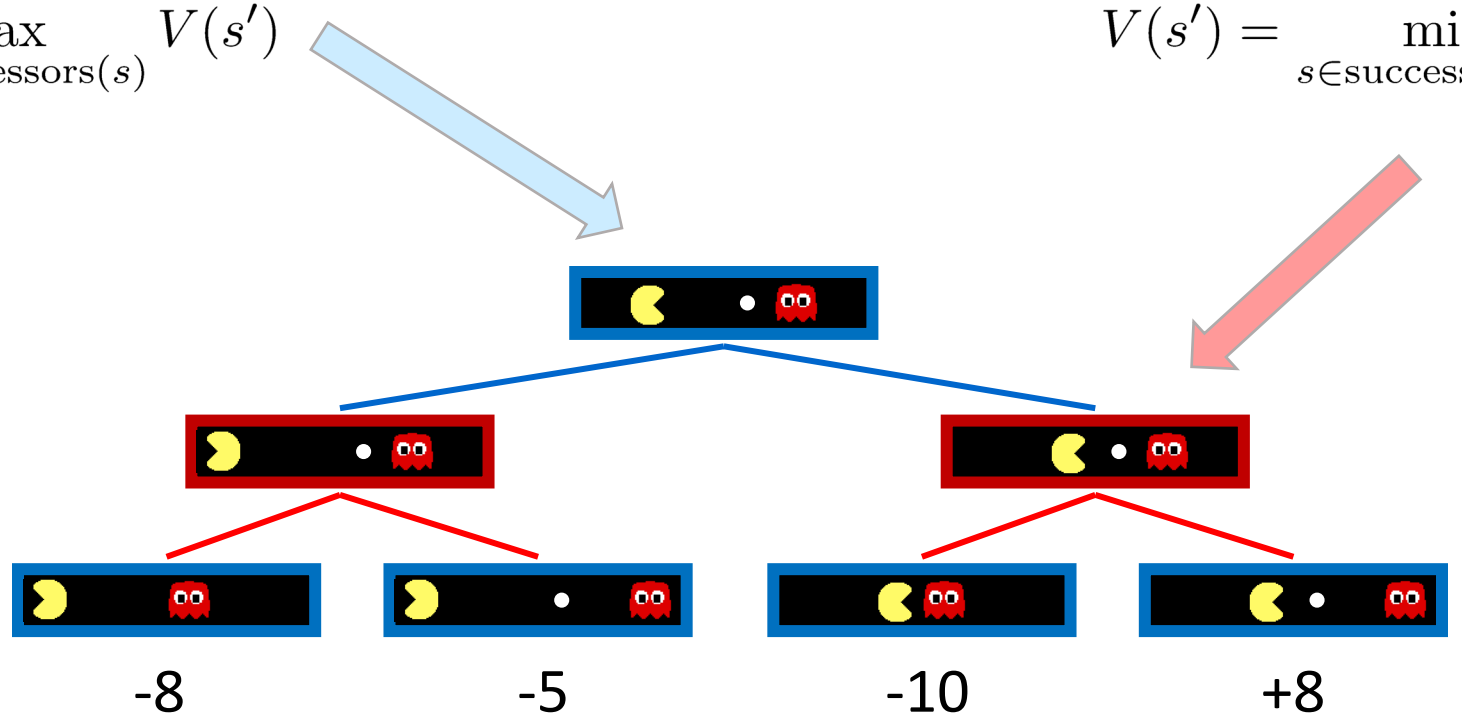
博弈搜索树

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

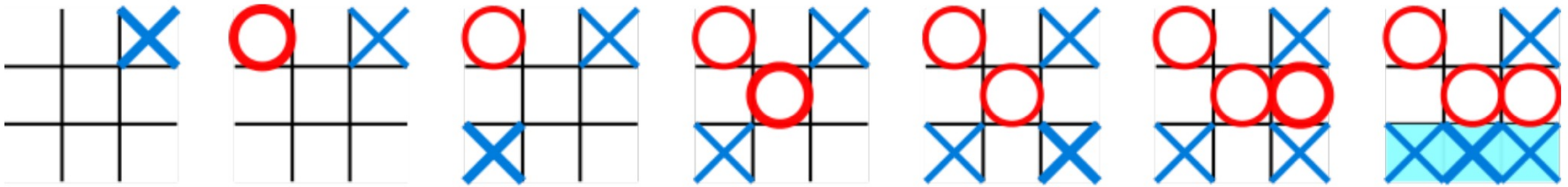


Terminal States:

$$V(s) = \text{known}$$

多步搜索

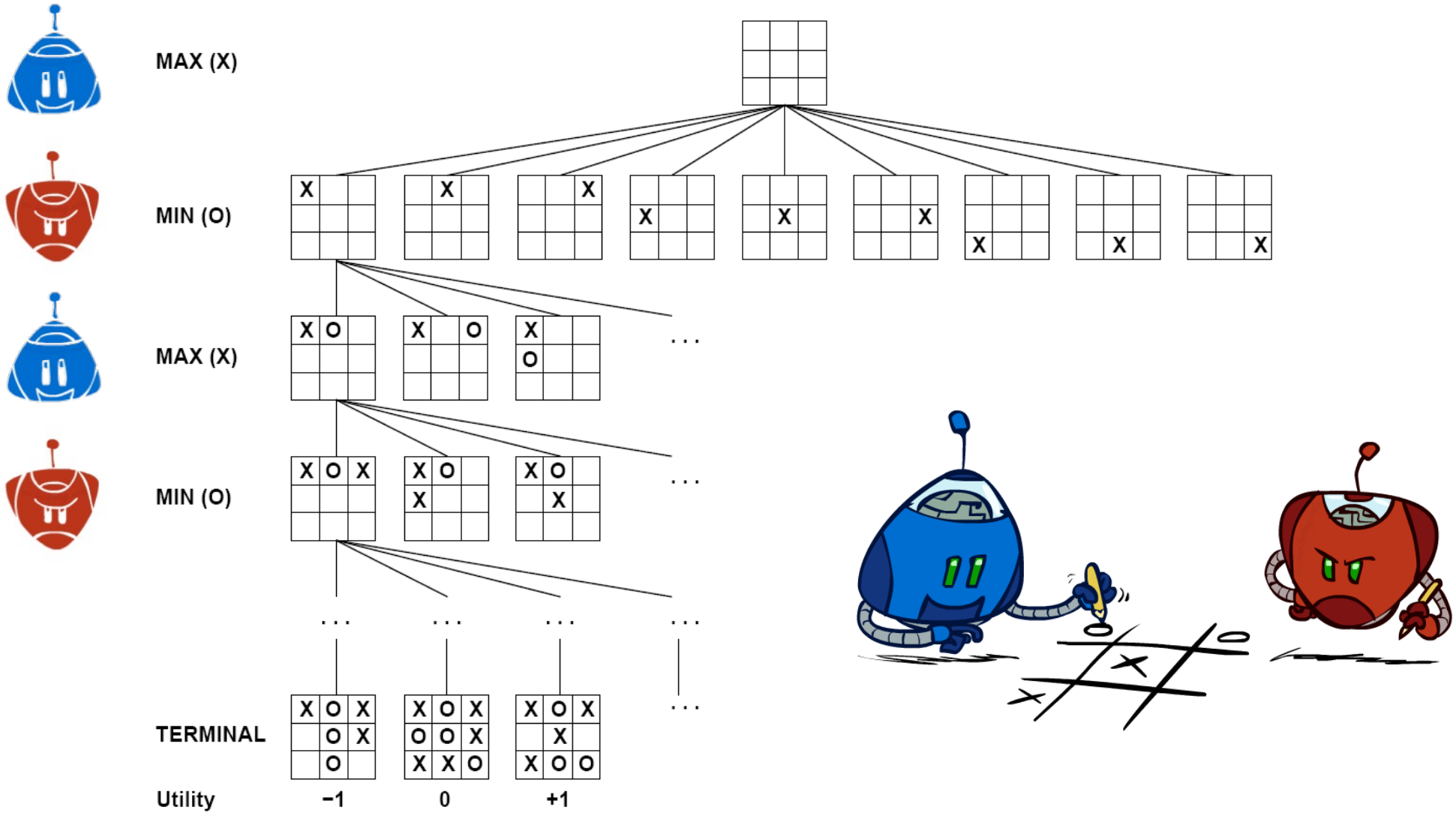
两人轮流在一有九格方盘上划加字或圆圈,谁先把三个同一记号排成横线、直线、斜线,即是胜者



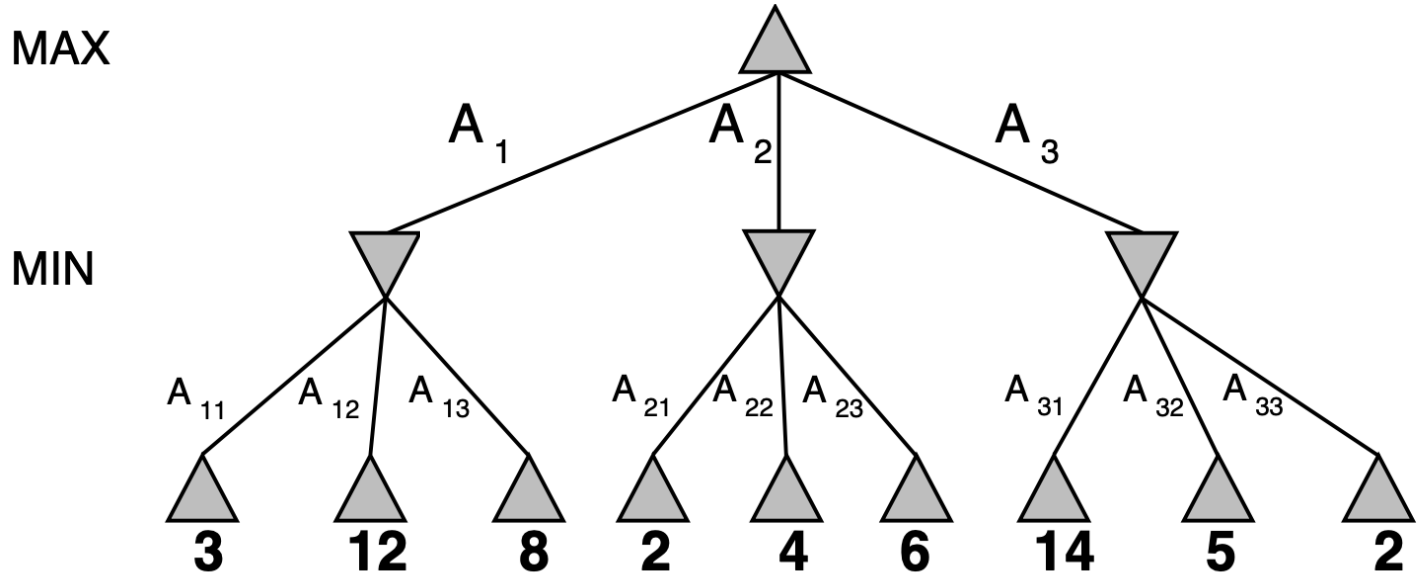
问题定义

- **状态**：状态s包括当前的游戏局面和当前行动的玩家
- **动作**：给定状态s，动作指的是$player(s)$在当前局面下可以采取的操作a，记动作集合为$actions(s)$
- **状态转移**：给定状态s和动作$a \in actions(s)$，状态转移函数$result(s,a)$决定了在s状态采取a动作后所得后继状态
- **终局状态检测**：终止状态检测函数$terminal_test(s)$用于测试游戏是否在状态s结束
- **终局得分**：终局得分$utility(s,p)$表示在终局状态s时玩家p的得分

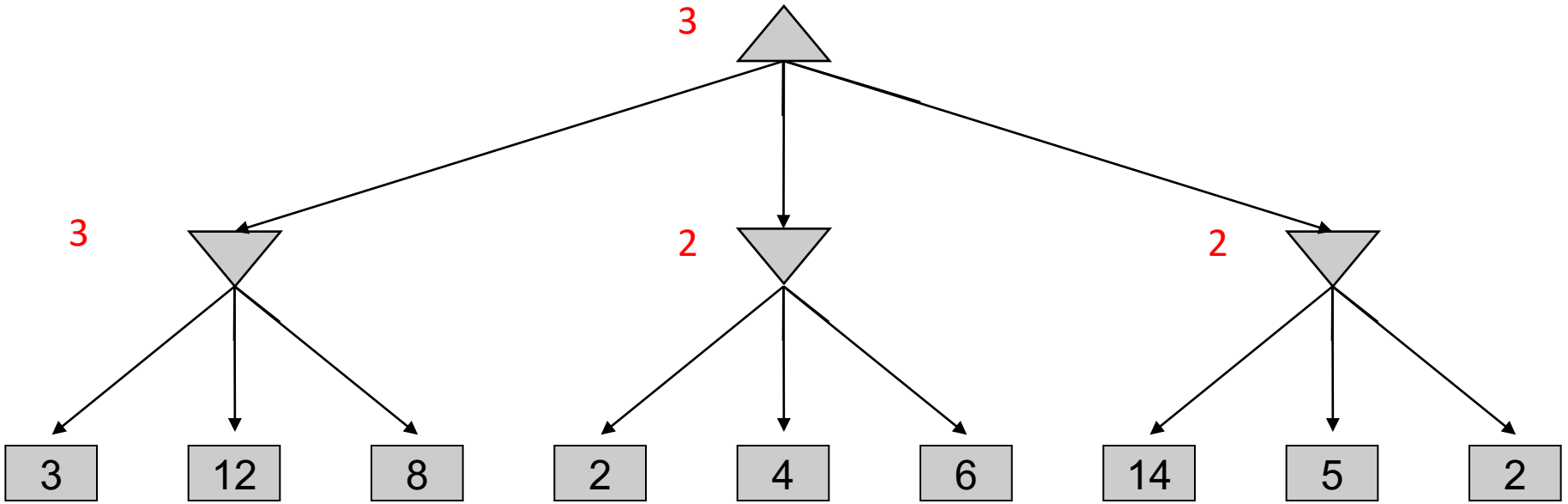
搜索树



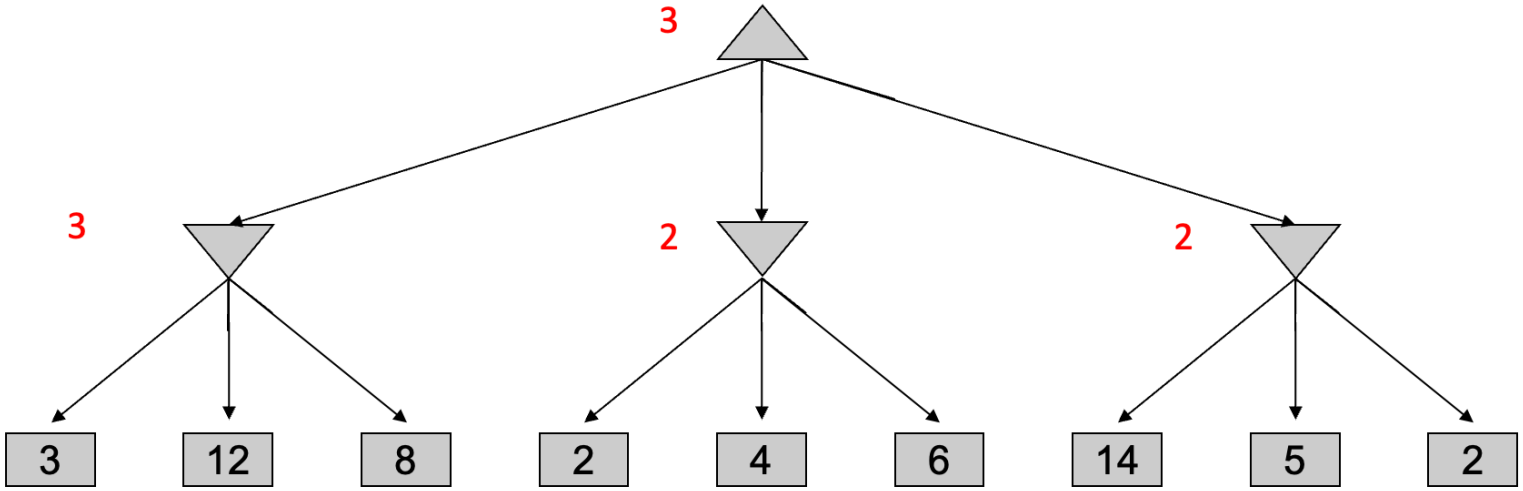
最优策略


MAX
MIN
A 1
A 2
A 3
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

最优策略



最优策略

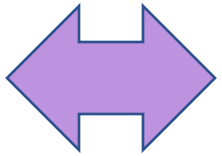


给定一棵博弈树，最优策略可以通过检查每个节点的极小极大值来决定，记为minimax(n)

$$\text{minimax}(s) = \begin{cases} \text{utility}(s), & \text{if terminal\_test}(s) \\ \max_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s,a)), & \text{if player}(s) = MAX \\ \min_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s,a)), & \text{if player}(s) = MIN \end{cases}$$

最小最大搜索

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```



```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

最小最大搜索

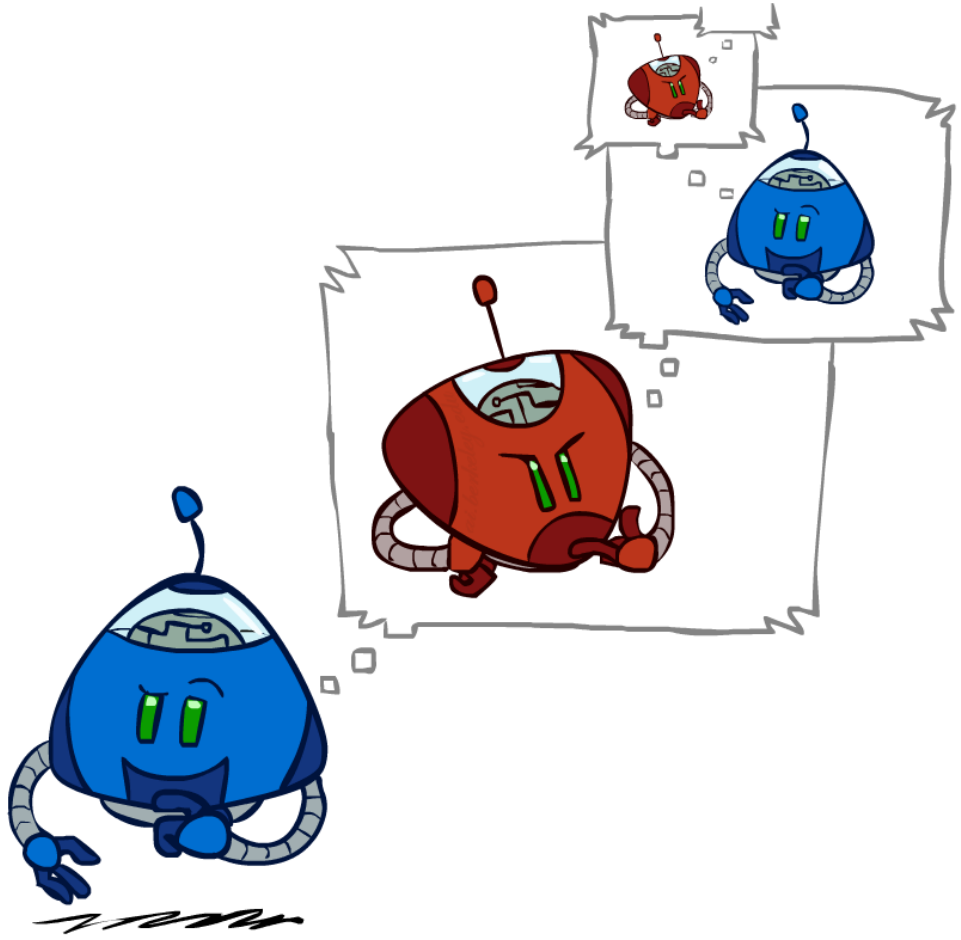
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the agent is MAX: return max-value(state)
    if the agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

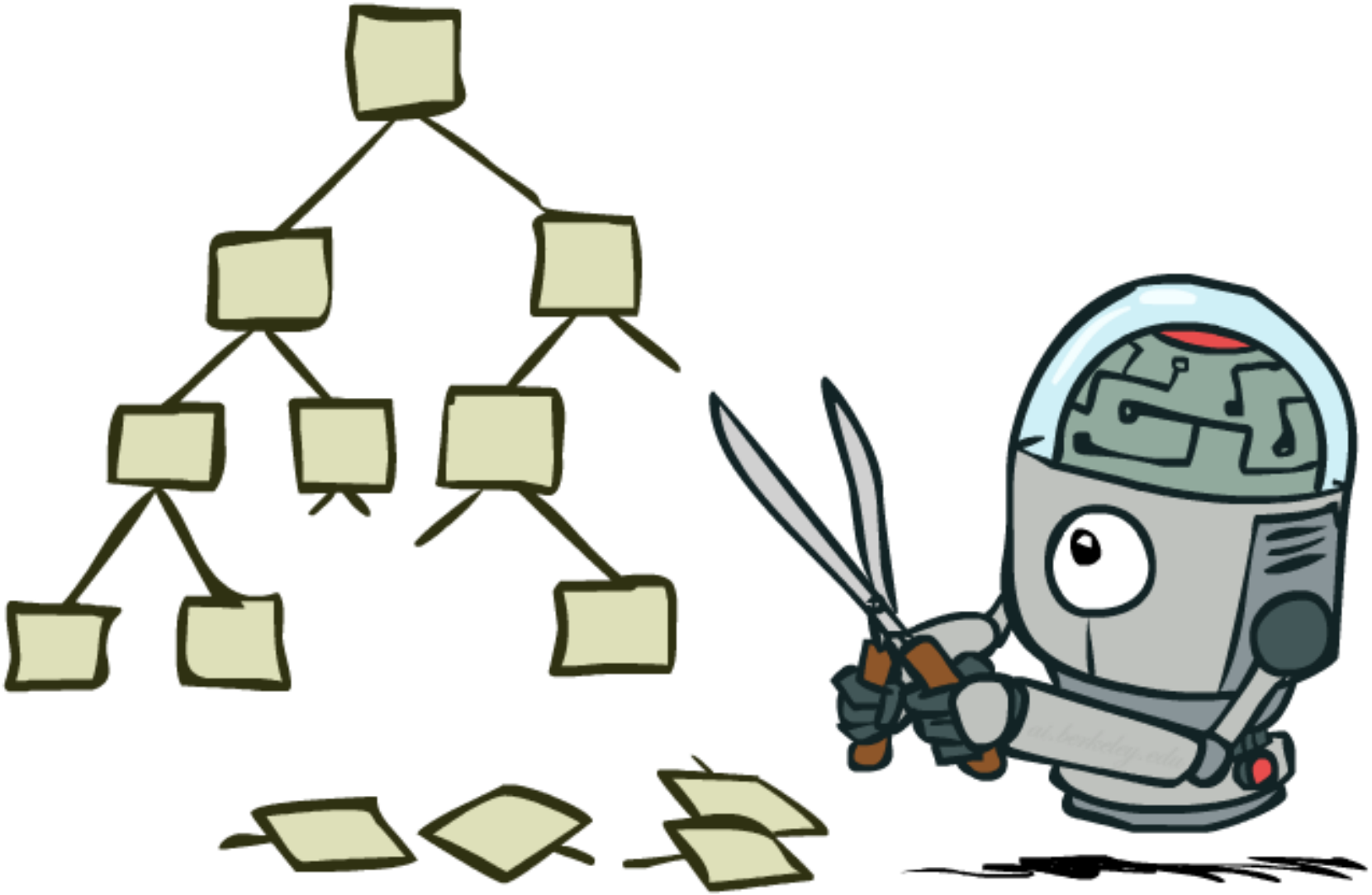
性能分析

- 时间和空间?
  - 和 DFS 类似
  - 时间复杂度: $O(b^m)$
  - 空间复杂度: O(bm)



- Example: For chess, $b \approx 35$, $m \approx 100$
  - 精确的搜索几乎是不可行的

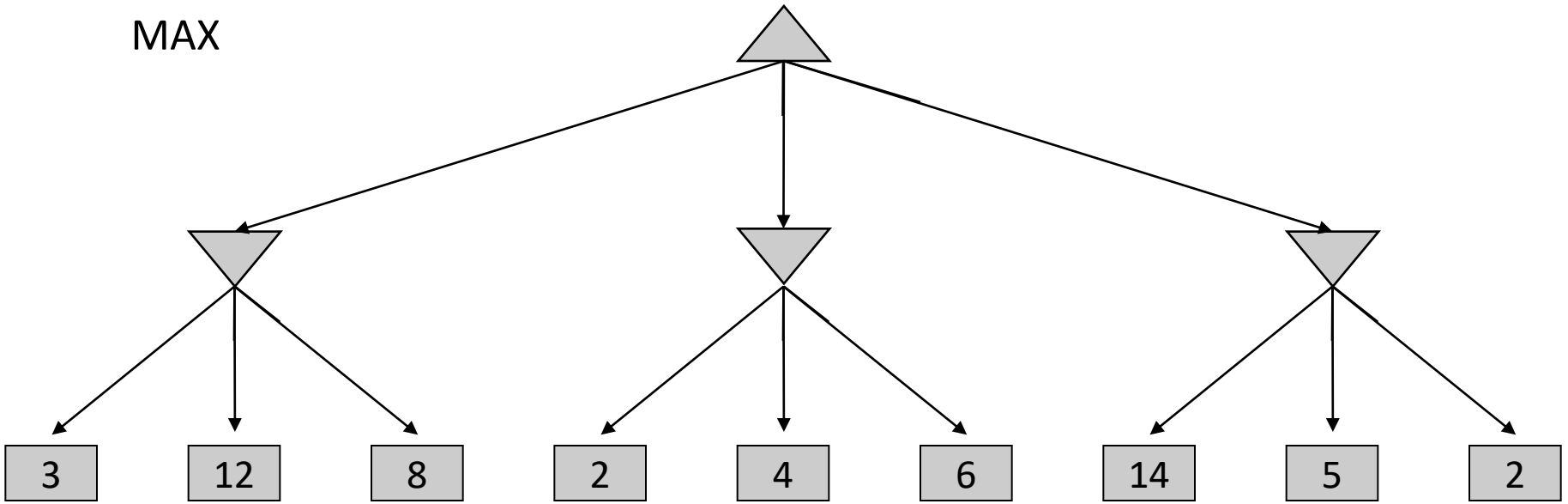
树剪枝



提纲

- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

树剪枝

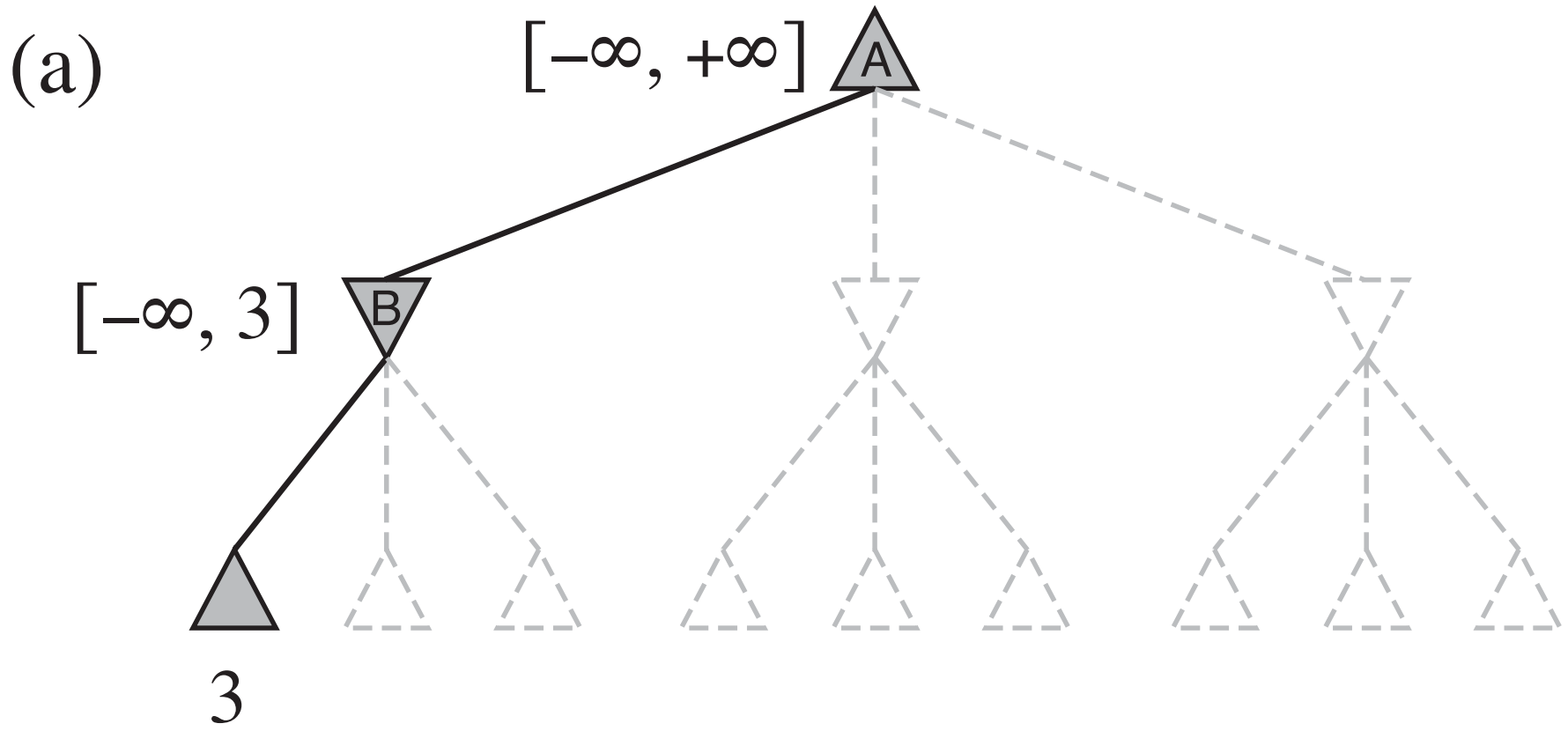


树剪枝

Vector:[alpha, beta]

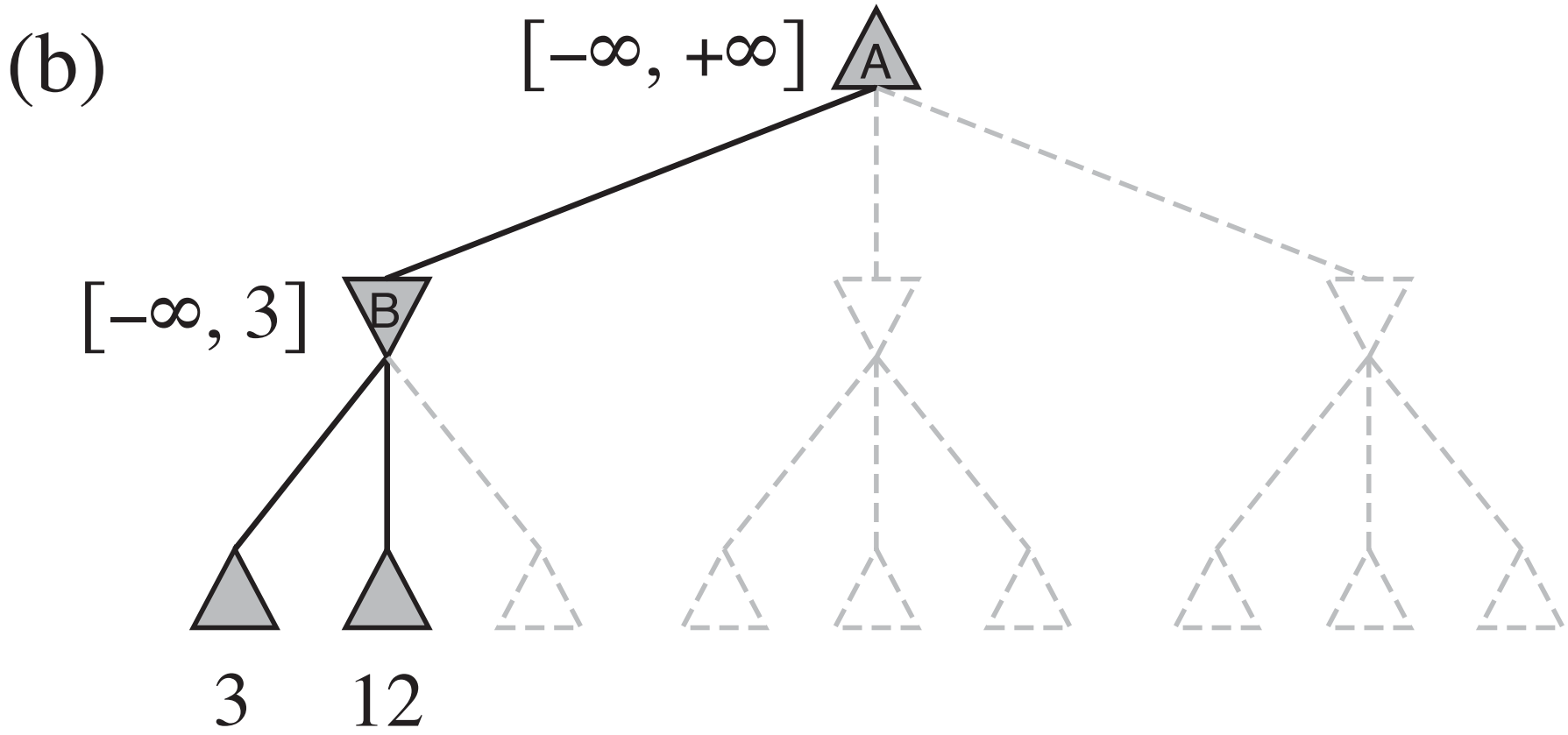
α: 目前为止Max玩家在当前路径上最高收益的下界(至少能拿这么多)，初始值为−∞

β: 目前为止Min玩家在当前路径上最低收益的上界(至多能让对手拿这么多)，初始值为+∞



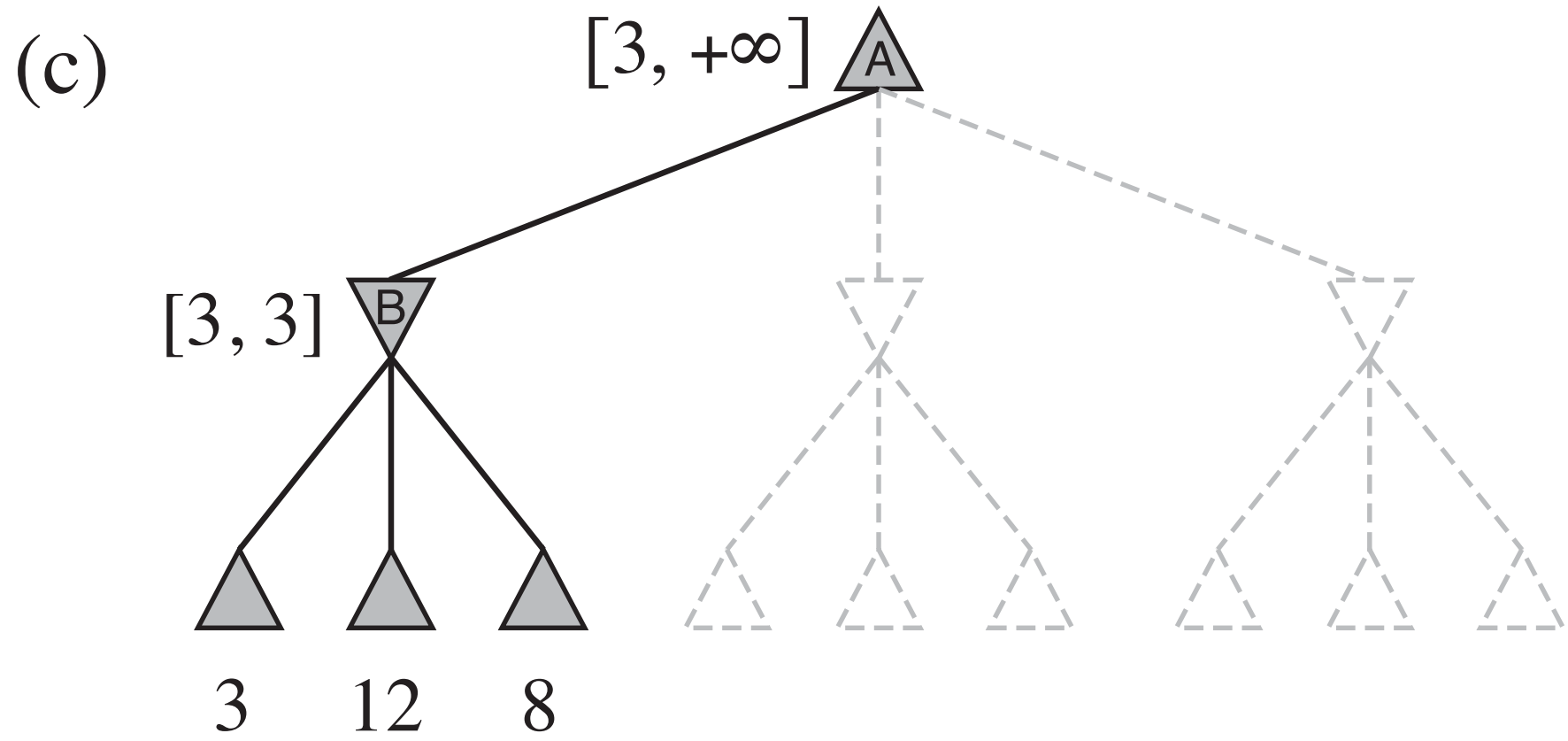
树剪枝

Vector:[alpha, beta]



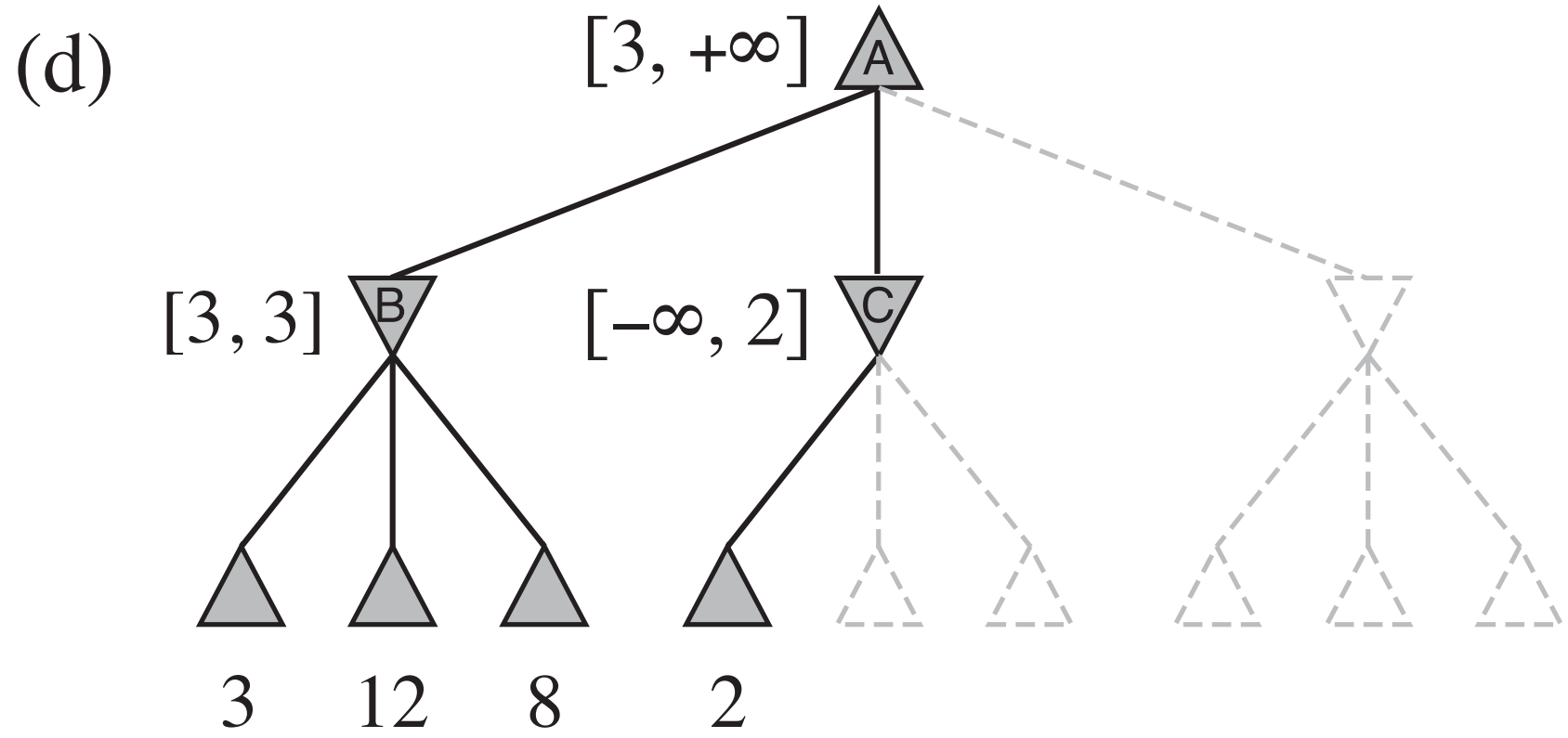
树剪枝

Vector:[alpha, beta]



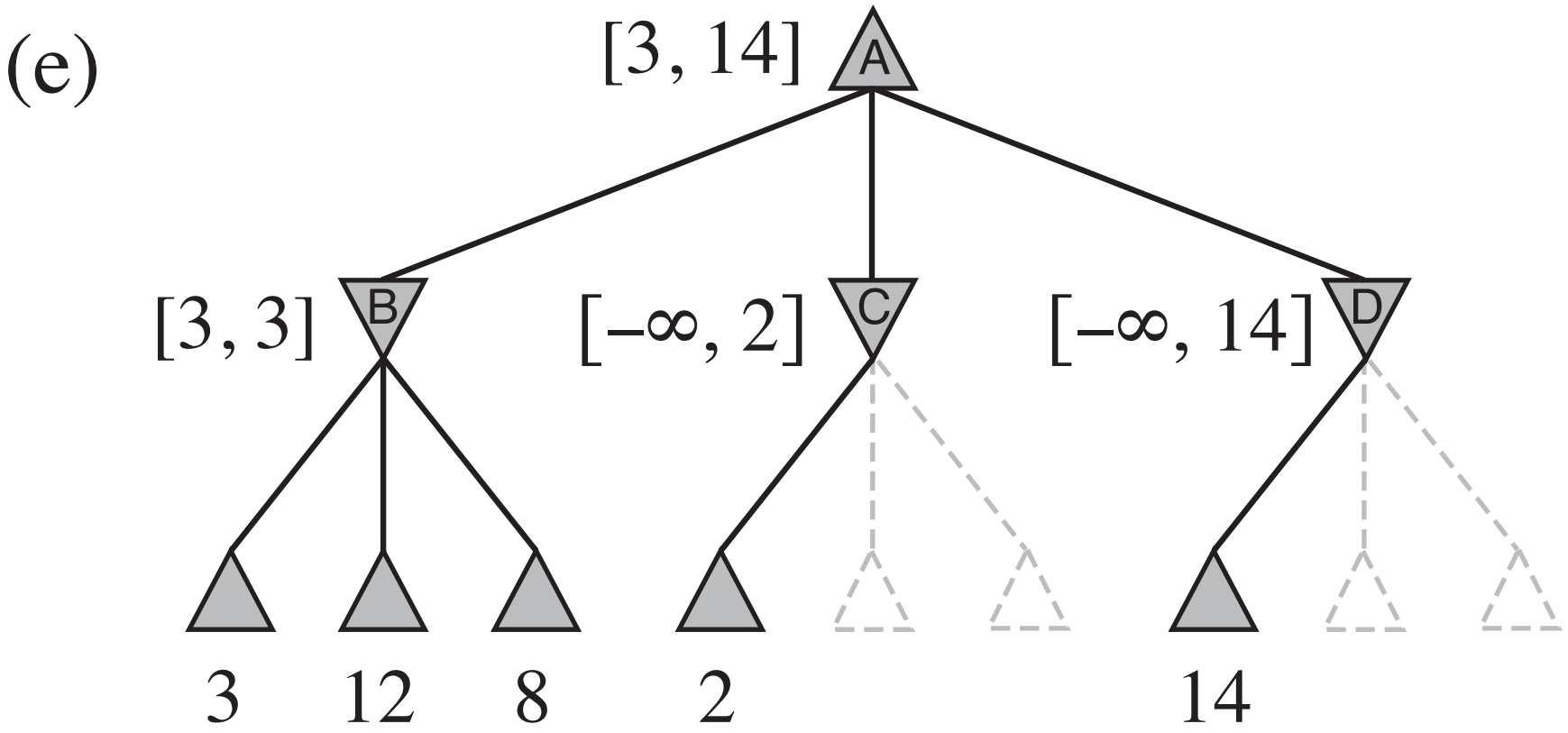
树剪枝

Vector:[alpha, beta]



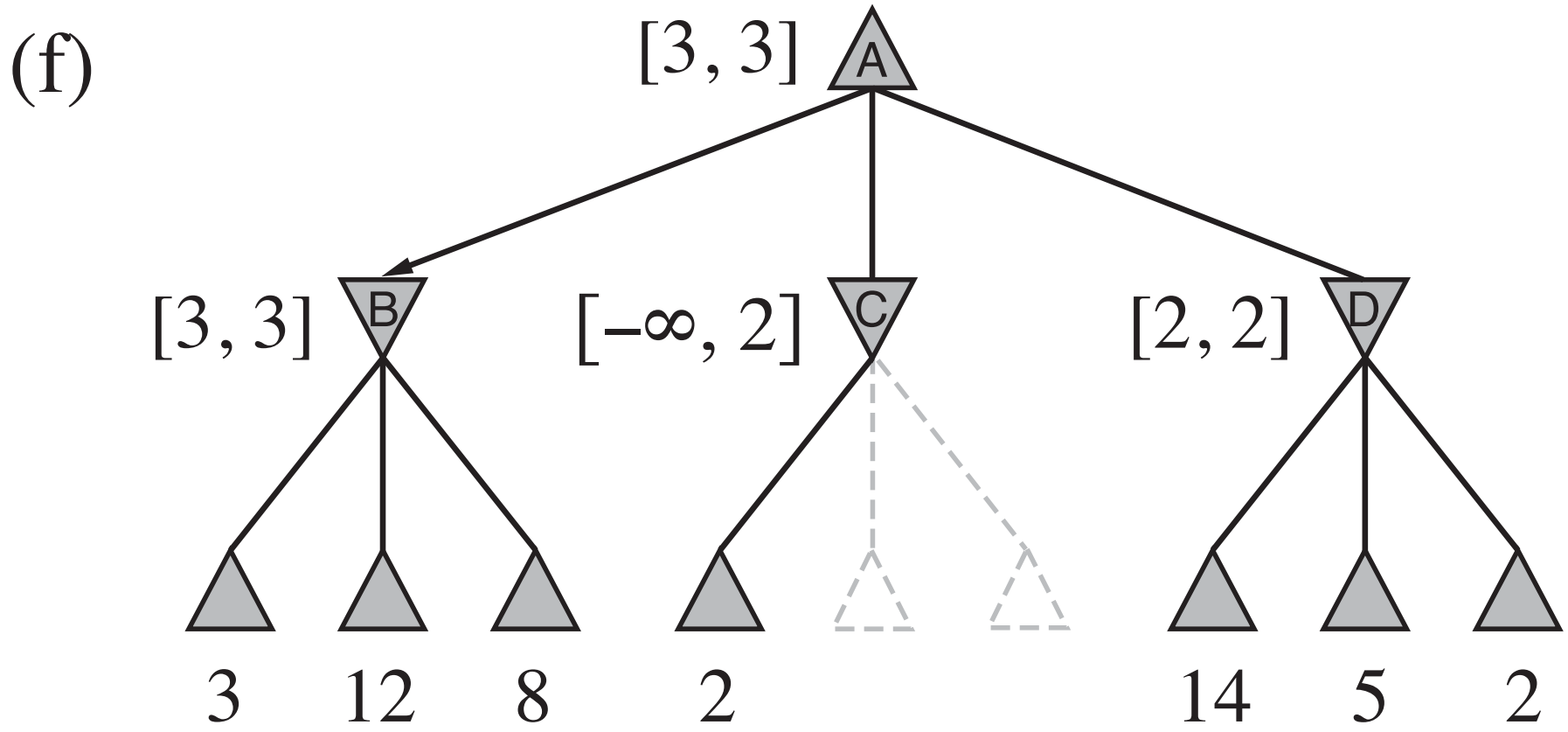
树剪枝

Vector:[alpha, beta]

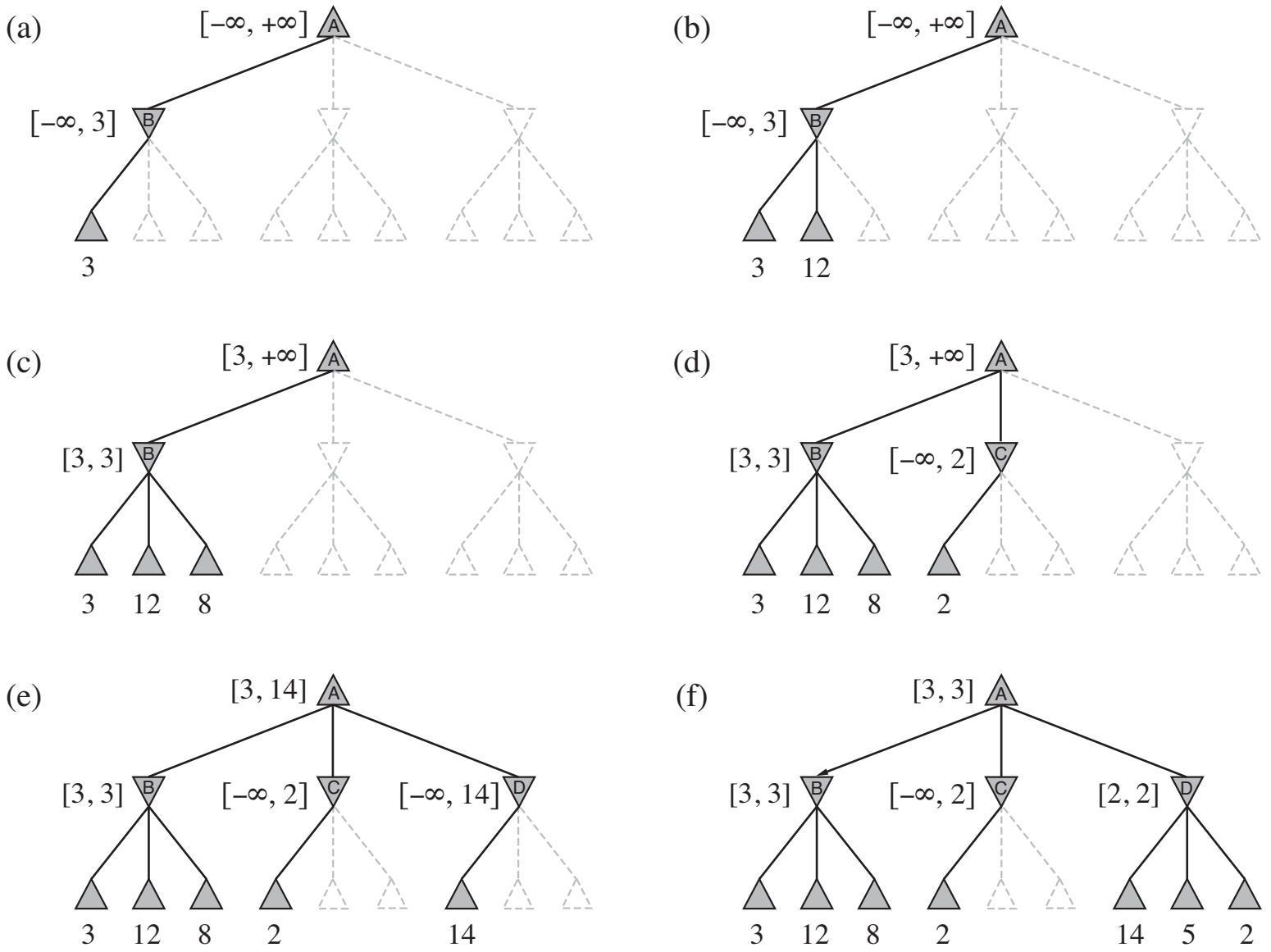


树剪枝

Vector:[alpha, beta]



树剪枝


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

Alpha-Beta剪枝

- 对于MAX节点，如果其孩子结点（MIN结点）的收益大于当前的α值，则将α值更新为该收益；

  对于MIN结点，如果其孩子结点（MAX结点）的收益小于当前的β值，则将β值更新为该收益。

  根结点（MAX结点）的α值和β值分别被初始化为$-\infty$和$+\infty$

- 随着搜索算法不断被执行，每个结点的α值和β值不断被更新。大体来说，每个结点的$[\alpha,\beta]$从其父结点提供的初始值开始，取值按照如下形式变化：α逐渐增加、β逐渐减少。不难验证，如果一个结点的α值和β值**满足$\alpha > \beta$**的条件，则该结点尚未被访问的后续结点就会被剪枝，因而不会被访问

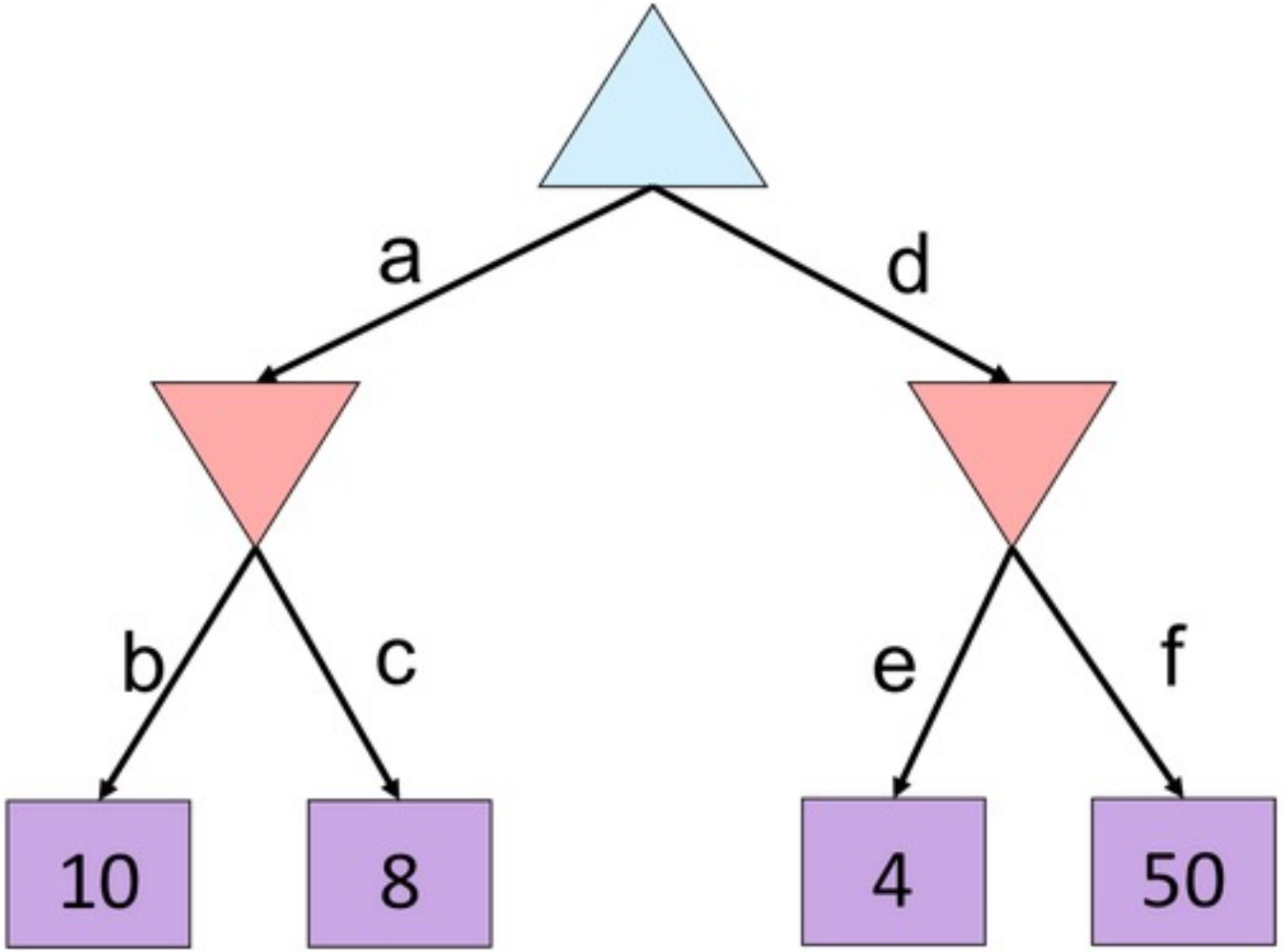
Alpha-Beta剪枝

α:目前为止路上发现的 MAX的最佳（即极大值）选择
β:目前为止路径上发现的MIN的最佳（即极小值）选择

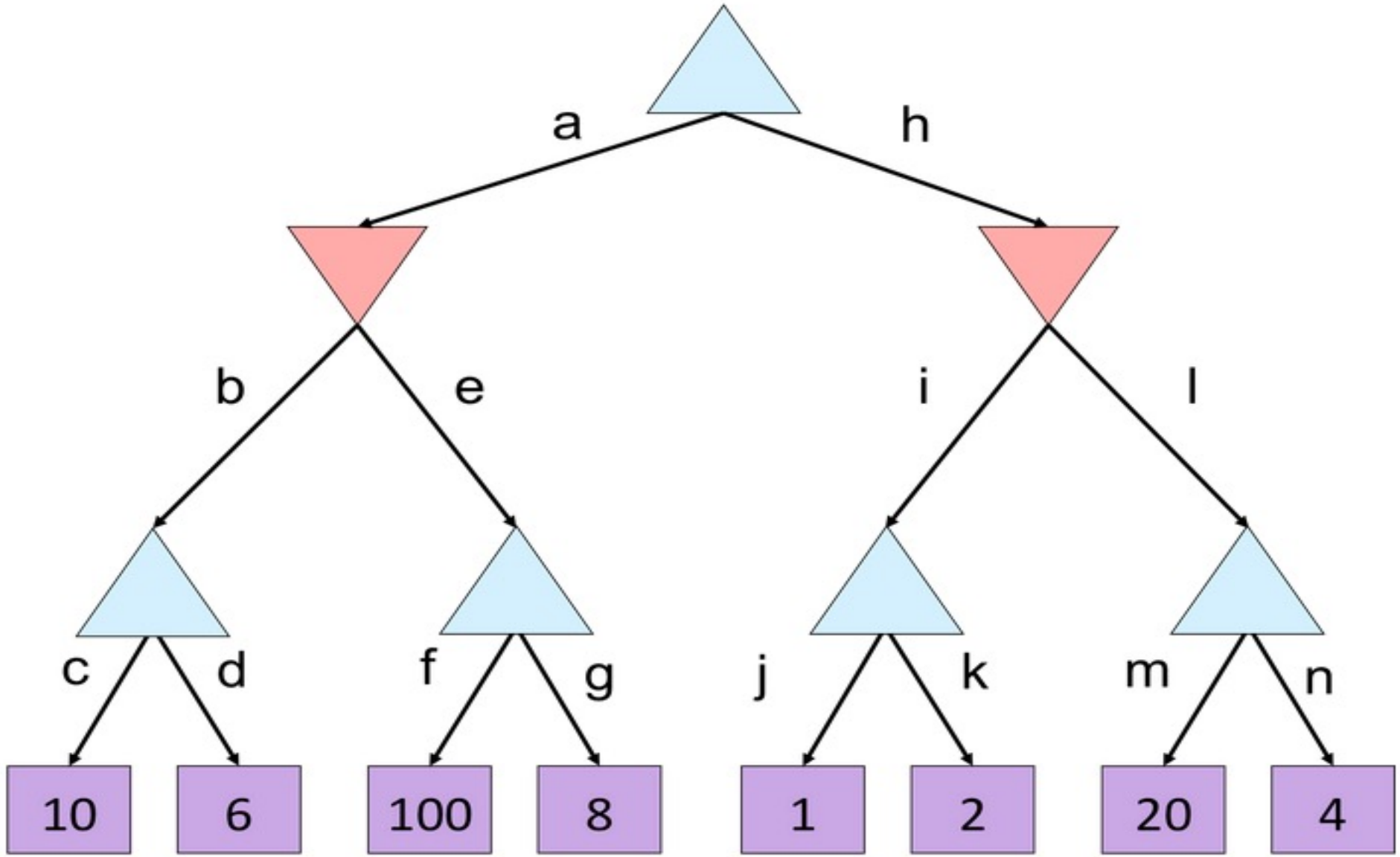
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

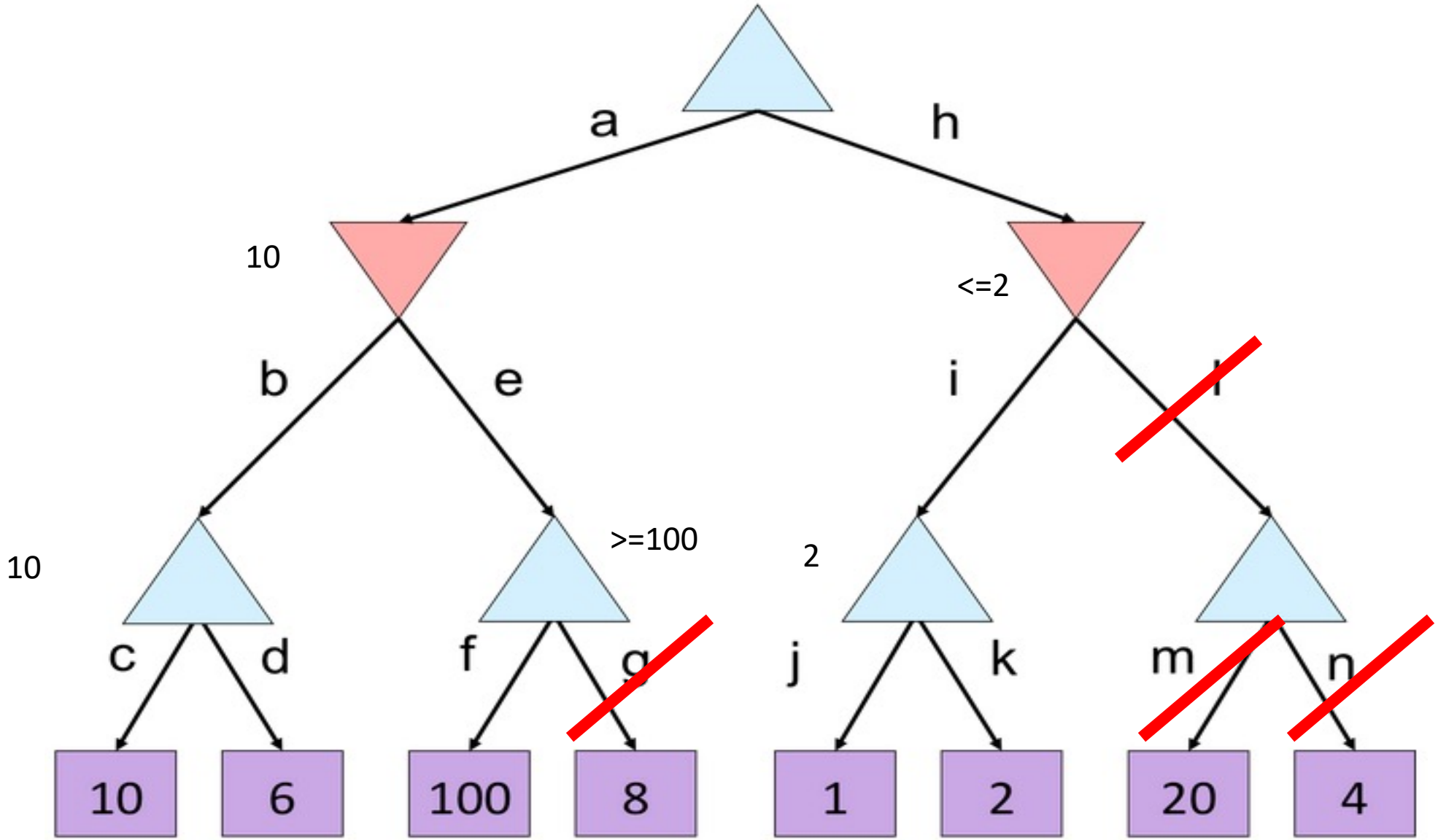
Quiz1: Alpha-Beta剪枝



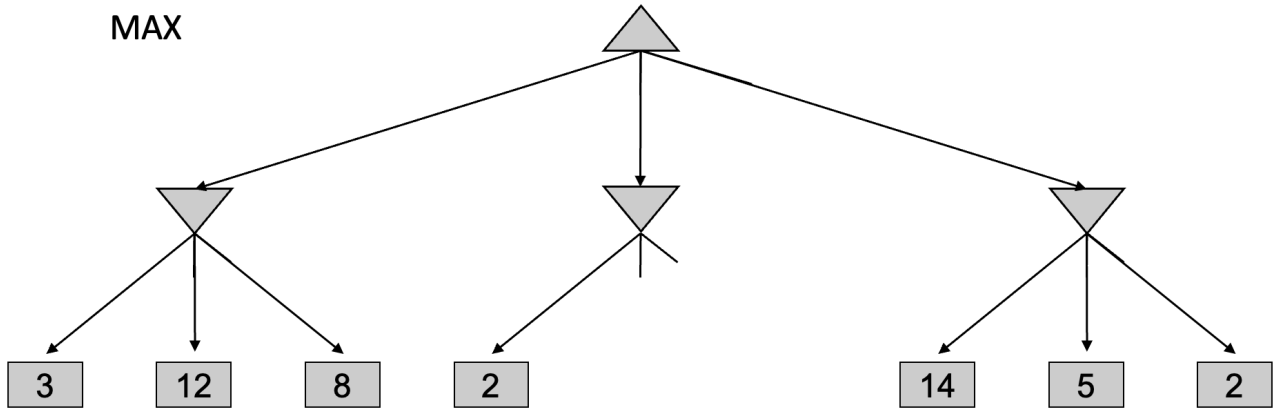
Quiz2: Alpha-Beta剪枝



Quiz2: Alpha-Beta剪枝



思考



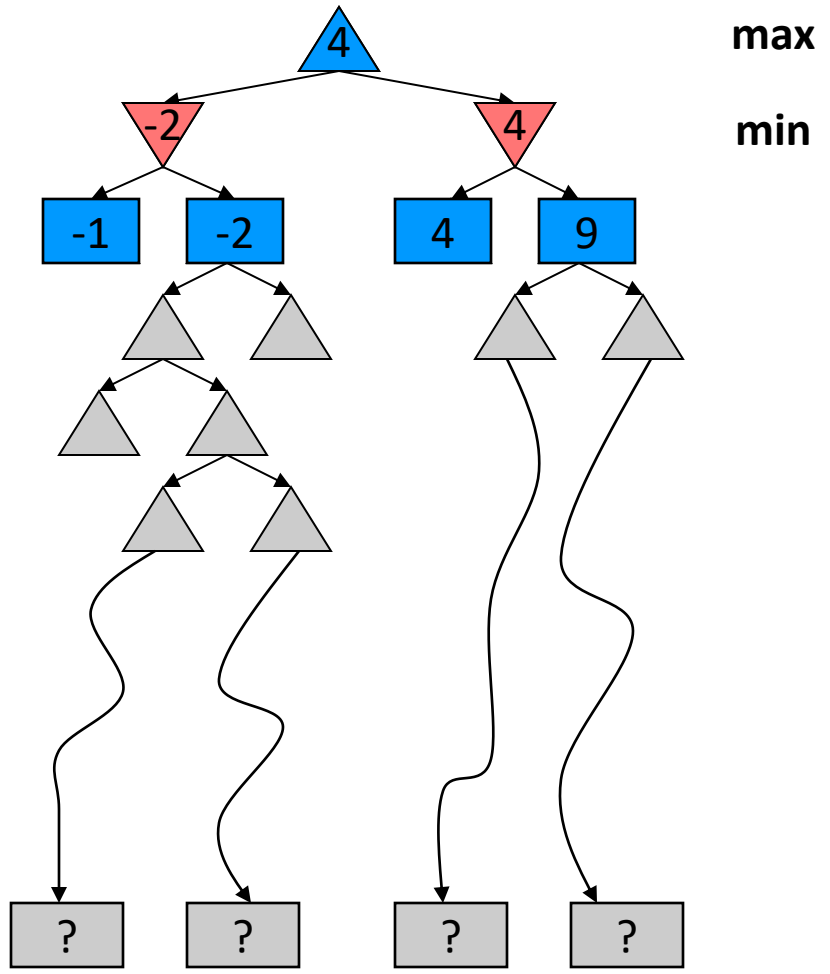
启发？　　搜索顺序很重要

可以设计方案对后继状态进行排序，

例如，对于象棋，可以设计排序规则：吃子>威胁>前进>后退

资源受限

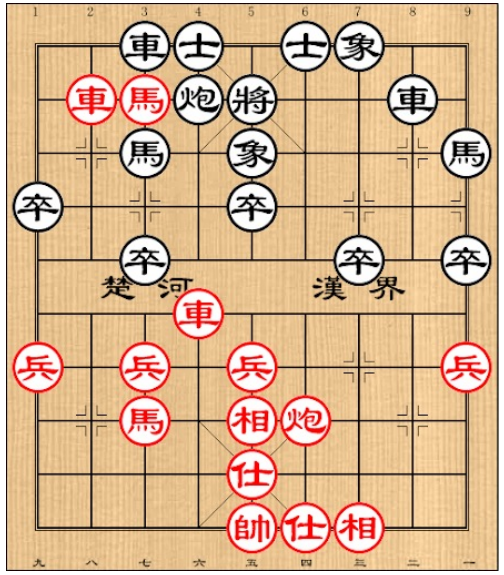
尽管alpha-beta剪枝能够避免搜索完整的空间，但是仍然要**搜索部分空间直至终止状态**，这样的搜索深度也是不现实的

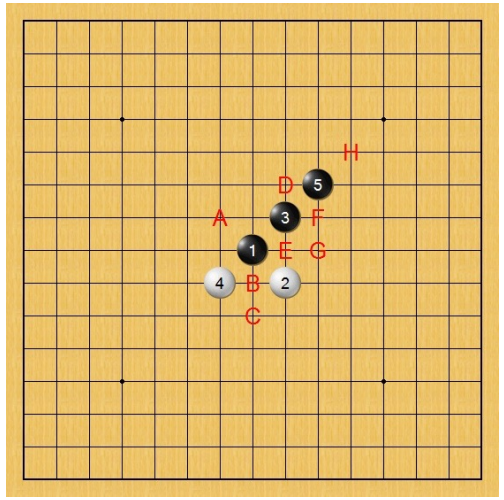


一个可行的思路：

参考启发式搜索，设计评估函数用于搜索中的状态，有效地把非终止节点变成终止节点

评估函数





如何设置评估函数？

以象棋为例，要考虑兵的数目、车的数目、马的数目等等

兵1分，马3分，车5分…

形式化描述：**加权线性函数**

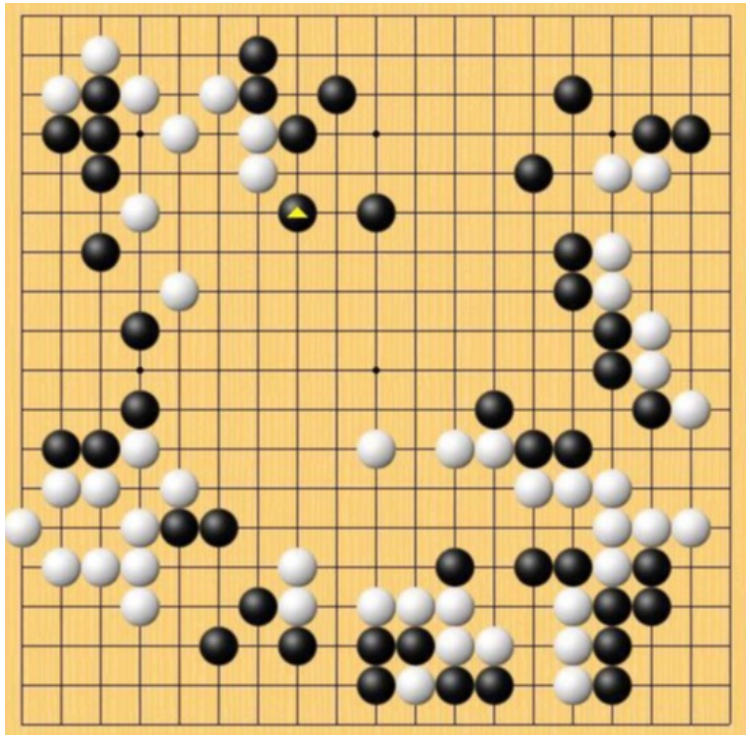
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

w_i表示权重，f_i是棋局的某个特征

评估函数

Alpha-beta搜索用于围棋

会面临什么挑战？



- **分支因子大**

围棋分支因子开始时为361，搜索层数受限

- **评估函数难设置**

现代围棋程序基本不采用alpha-beta搜索